```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim, double lambda);

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, color_img;

    // adjustment: if number of arguments is greater than 3, since third will be lambda value
    if ( argc > 3 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    // declare 1D array of double pointers filter
    double* filter[5];
    // declare double to store scaling factor lambda - default value below
    double lambda = 1.0;
    if (argc > 2) {
```

```
48          lambda = atof(argv[2]);
49      }
50
51      // iterate through 1D array filter and allocate enough memory for 9
          doubles each
52      for (int i = 0; i < 5; i++) {
53          filter[i] = malloc(sizeof(double) * 9);
54      }
55      // populate filter array according to g(m,n)
56      for (int i = 0; i < 5; i++) {
57          for (int j = 0; j < 5; j++) {
58              if (i == 2 && j == 2) {
59                  filter[i][j] = 1 + lambda * (1 - 1.0 / 25.0);
60              }
61              else {
62                  filter[i][j] = lambda * (-1.0 / 25.0);
63              }
64          }
65      }
66
67      /* set up structure for output color image */
68      /* Note that the type is 'c' rather than 'g' */
69      get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
70
71      // declare and initialize integer to store the dimension of the point
          spread function
72      int PSF_dim = 5;
73
74      // apply filter using applyFilter function as defined below main
75      applyFilter(&color_img, &input_img, filter, PSF_dim, lambda);
76
77      /* open color image file */
78      if ( ( fp = fopen ( "sharpened.tif", "wb" ) ) == NULL ) {
79          fprintf ( stderr, "cannot open file color.tif\n");
80          exit ( 1 );
81      }
82
83      /* write color image */
84      if ( write_TIFF ( fp, &color_img ) ) {
85          fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
86          exit ( 1 );
87      }
88
89      /* close color image file */
90      fclose ( fp );
91
92      /* de-allocate space which was used for the images */
93      free_TIFF ( &(input_img) );
94      free_TIFF ( &(color_img) );
```

```
 95
 96      return(0);
 97  }
 98
 99  void error(char *name)
100  {
101      printf("usage:  %s  image.tiff \n\n",name);
102      printf("this program reads in a 24-bit color TIFF image.\n");
103      printf("It then horizontally filters the green component, adds noise,
           \n");
104      printf("and writes out the result as an 8-bit image\n");
105      printf("with the name 'green.tiff'.\n");
106      printf("It also generates an 8-bit color image,\n");
107      printf("that swaps red and green components from the input image");
108      exit(1);
109  }
110
111  // limitIntensity function definition
112  int limitIntensity(double inputValue) {
113      // declare an integer variable newValue and initialize it to zero
114      int newValue = 0;
115      // if input value parameter is less than zero, assign new value to 0
116      if (inputValue < 0) {
117          newValue = 0;
118      }
119      // if input value parameter is greater than 255, assign new value to
           255
120      else if(inputValue > 255) {
121          newValue = 255;
122      }
123      // otherwise, assign new value to the input value parameter re-cast as
           an integer
124      else {
125          newValue = (int)inputValue;
126      }
127      return newValue;
128  }
129
130  // applyFilter function definition
131  void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img,
       double **usedFilter, int PSF_dim, double lambda) {
132      // declare and define N - the dimension of the point spread function
           (PSF)
133      int N = (PSF_dim - 1) / 2;
134      // declare and define image height and width based on input image TIFF
           struct methods
135      int img_height = input_img->height;
136      int img_width = input_img->width;
137      // declare doubles to store red, green, and blue value for each pixel
```

```
138     double redPlane, greenPlane, bluePlane;
139     // declare PSF variables
140     int m, n;
141     // declare variables to store current location within PSF
142     int a, b;
143     // for each pixel
144     for (int i = 0; i < img_height; i++) {
145         for (int j = 0; j < img_width; j++) {
146             // initialize RGB values to zero
147             redPlane = 0.0;
148             greenPlane = 0.0;
149             bluePlane = 0.0;
150             // for each pixel in the PSF (5*5 in this case)
151             for (m = -N; m <= N; m++) {
152                 for (n = -N; n <= N; n++) {
153                     // assign a and b to current PSF matrix location
154                     a = i - m;
155                     b = j - n;
156                     // if a and b are within the image boundaries
157                     if (a >= 0 && a < img_height && b >= 0 && b <
                    img_width) {
158                         // apply filter by summing across PSF according to
                     difference equation for 2D filters
159                         redPlane += usedFilter[m + N][n + N] * input_img-
                    >color[0][a][b];
160                         greenPlane += usedFilter[m + N][n + N] *
                    input_img->color[1][a][b];
161                         bluePlane += usedFilter[m + N][n + N] * input_img-
                    >color[2][a][b];
162                     }
163                 }
164 
165             }
166             // populate output image method for color after calling
                  limitIntensity function to ensure acceptable RGB values
167             output_img->color[0][i][j] = limitIntensity(redPlane);
168             output_img->color[1][i][j] = limitIntensity(greenPlane);
169             output_img->color[2][i][j] = limitIntensity(bluePlane);
170         }
171     }
172 }
```